Gallager Codes – Recent Results

David J. C. MacKay (mackay@mrao.cam.ac.uk)
Cavendish Laboratory, Cambridge, CB3 0HE, United Kingdom.

Abstract

In 1948, Claude Shannon posed and solved one of the fundamental problems of information theory. The question was whether it is possible to communicate reliably over noisy channels, and, if so, at what rate. He defined a theoretical limit, now known as the Shannon limit, up to which communication is possible, and beyond which communication is not possible. Since 1948, coding theorists have attempted to design error-correcting codes capable of getting close to the Shannon limit.

In the last decade remarkable progress has been made using codes that are defined in terms of sparse random graphs, and which are decoded by a simple probability-based message-passing algorithm.

This paper reviews low–density parity–check codes (Gallager codes), repeat-accumulate codes, and turbo codes, emphasising recent advances. Some previously unpublished results are then presented, describing (a) experiments on Gallager codes with small blocklengths; (b) a stopping rule for decoding of repeat-accumulate codes, which saves computer time and allows block decoding errors to be detected and flagged; and (c) the empirical power-laws obeyed by decoding times of sparse graph codes.

1 Introduction

The central problem of communication theory is to construct an encoding and a decoding system that make it possible to communicate reliably over a noisy channel. The encoding system uses the source data to select a codeword from a set of codewords. The decoding algorithm ideally infers, given the output of the channel, which codeword in the code is the most likely to have been transmitted; for an appropriate definition of distance, this is the 'closest' codeword to the received signal. A good code is one in which the codewords are well spaced apart, so that codewords are unlikely to be confused.

Designing a good and practical error correcting code is difficult because (a) it is hard to find an explicit set of well-spaced codewords; and (b) for a generic code, decoding, *i.e.*, finding the closest codeword to a received signal, is intractable.

However, a simple method for designing codes, first pioneered by Gallager (1962), has recently been rediscovered and generalized. The codes are defined in terms of sparse random graphs. Because the graphs are constructed randomly, the codes are likely to have well-spaced codewords. And because the codes' constraints are defined by a sparse graph, the decoding problem can be solved – almost optimally – by message-passing on the graph. The practical performance of Gallager's codes and their modern cousins is vastly better than the performance of the codes with which textbooks have been filled in the intervening years.

2 Sparse Graph Codes

In a **sparse graph code**, the nodes in the graph represent the transmitted bits and the constraints they satisfy. For a linear code with a codeword length N and rate $R = K/N$, the number of constraints is of order $M = N - K$. [There could be more constraints, if they happen to be redundant.] Any linear code can be described by a graph, but what makes a sparse graph code special is that each constraint involves only a small number of variables in the graph: the number of edges in the graph scales roughly linearly with N, rather than as N^2.

The graph defining a **low–density parity–check code**, or Gallager code (Gallager 1962; Gallager 1963; MacKay 1999), contains two types of node: codeword bits, and parity constraints. In a regular (j, k) Gallager code (figure 1a), each codeword bit is connected to j parity constraints and each constraint is connected to k bits. The connections in the graph are made at random.

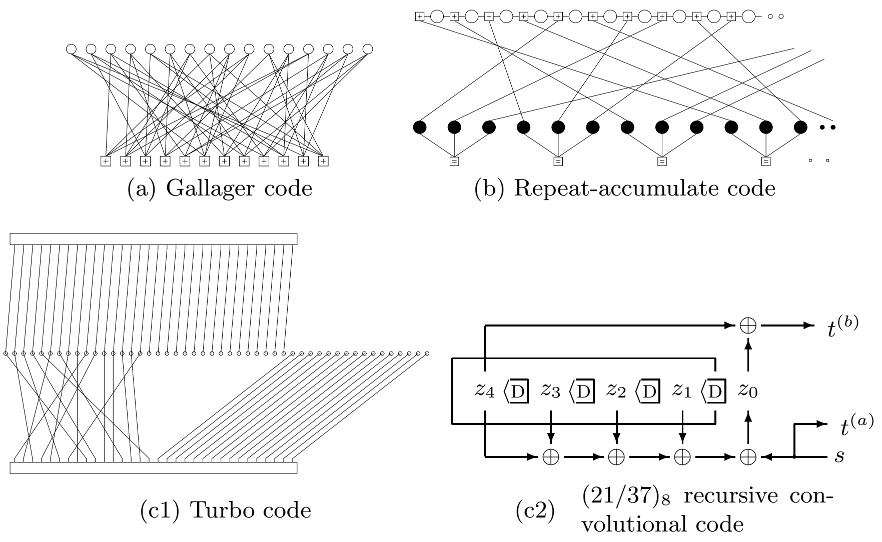
Repeat-accumulate codes (Divsalar *et al.* 1998) can be represented by a graph with four types of node (figure 1b): equality constraints $\boxed{=}$, intermediate binary variables (black circles), parity constraints $\boxed{+}$, and the transmitted bits (white circles). The encoder sets each group of intermediate bits to values read from the source. These bits are put through a fixed random permutation. The transmitted stream is the accumulated sum (modulo 2) of the permuted intermediate bits.

In a **turbo code** (Berrou and Glavieux 1996), the K source bits drive two linear feedback shift registers, which emit parity bits (figure 1c).

All these codes can be decoded by a local message-passing algorithm on the graph, the sum-product algorithm (MacKay and Neal 1996; McEliece *et al.* 1998), and, while this algorithm is not the optimal decoder, the empirical results are record-breaking. Figure 2 shows the performance of various sparse graph codes on a Gaussian channel. In figure 2(a) turbo codes with rate 1/4 are compared with regular and irregular Gallager codes over GF(2), GF(8) and GF(16). In figure 2(b) the performance of repeat-accumulate codes of various blocklengths and rate 1/3 is shown.

The best sparse graph codes

Which of the three types of sparse graph code is 'best' depends on the chosen rate and blocklength, the permitted encoding and decoding complexity, and



(a) Gallager code

(b) Repeat-accumulate code

(c1) Turbo code

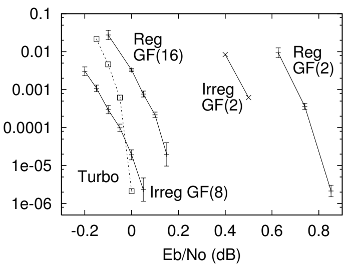
(c2) $(21/37)_8$ recursive convolutional code

Figure 1. Graphs of three sparse graph codes.

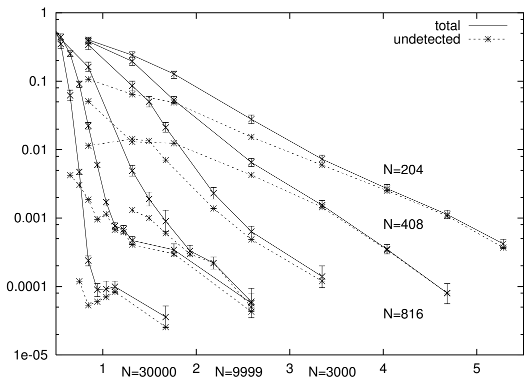
(a) A rate 1/4 low–density parity–check code (Gallager code) with blocklength $N = 16$, and $M = 12$ constraints. Each white circle represents a transmitted bit. Each bit participates in $j = 3$ constraints, represented by $\boxed{+}$ squares. Each $\boxed{+}$ constraint forces the sum of the $k = 4$ bits to which it is connected to be even.

(b) A repeat-accumulate code with rate 1/3. Each white circle represents a transmitted bit. Each black circle represents an intermediate binary variable. Each $\boxed{=}$ constraint forces the variables to which it is connected to be equal.

(c) A turbo code with rate 1/3. (c1) The circles represent the codeword bits. The two rectangles represent rate 1/2 convolutional codes (c2), with the systematic bits $\{t^{(a)}\}$ occupying the left half of the rectangle and the parity bits $\{t^{(b)}\}$ occupying the right half.



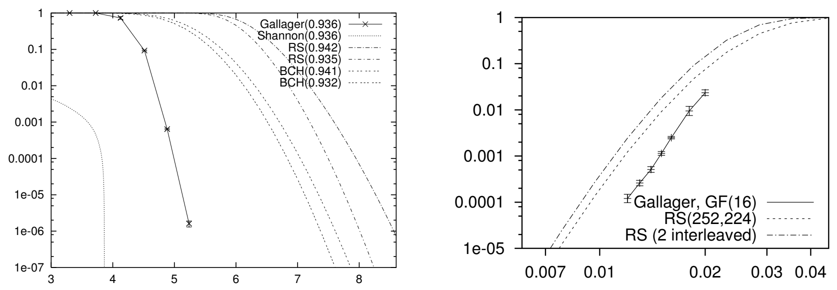
(a)



(b)

Figure 2. (a) Bit error probabilities for communication over a Gaussian channel at rate 1/4: left–right : Irregular LDPC, $GF(8)$, transmitted blocklength 24000 bits; JPL turbo, $N = 65536$ bits (dotted line); Regular LDPC, $GF(16)$, $N = 24448$ bits; Irregular LDPC, $GF(2)$, $N = 64000$ bits; Regular LDPC, $GF(2)$, $N = 40000$ bits. [From Davey and MacKay (1998).]

(b) Block error probability of repeat-accumulate codes with rate 1/3 and various blocklengths, versus E_b/N_0. The dotted lines show the frequency of undetected errors.



(a) (b)

Figure 3. (a) A regular binary Gallager code with column weight $j = 4$, rate $R = 0.936$ and blocklength $N = 4376$ ($K = 4094$), compared with BCH codes and interleaved Reed-Solomon codes with similar rates, on a Gaussian channel. Hard-input bounded-distance decoding is assumed for the BCH and RS codes. Vertical axis: block error probability. Horizontal axis: E_b/N_0 [Curves that are further to the left are best.] (b) A Gallager code over GF(16), rate 8/9, blocklength $N = 3996$ bits, applied to a 16-ary symmetric channel, and compared with interleaved RS codes with similar rates. Vertical axis: block error probability. Horizontal axis: channel symbol error probability. [Curves that are further to the right are best.] From MacKay and Davey (1998).

the question of whether occasional undetected errors are acceptable (turbo codes and repeat-accumulate codes both typically make occasional undetected errors, even at high signal-to-noise ratios, because they have a small number of low weight codewords; Gallager codes do not typically show such an error floor).

Gallager codes are the most versatile; it's easy to make a competitive Gallager code with almost any rate and blocklength, as is illustrated in figure 3. Figure 3(a) shows the performance of a high-rate regular binary Gallager code; it outperforms BCH codes and Reed-Solomon codes on this channel. And figure 3(b) shows the performance of a high rate Gallager code over GF(16) on a 16-ary symmetric channel: even though this channel is the sort of channel for which Reed-Solomon codes are intended, the Gallager code still manages to perform a little better than the RS code.

The best binary Gallager codes found so far are *irregular* codes whose parity check matrices have nonuniform weight per column (Luby *et al.* 1998; Urbanke *et al.* 1999). The carefully constructed codes of Urbanke, Richardson and Shokrollahi outperform turbo codes at blocklengths longer than 10^4 bits, with especially impressive results at 10^6 bits, where a rate 1/2 irregular Gallager code has a bit error probability of 10^{-6} at just 0.13 dB from capacity, beating comparable turbo codes by more than 0.3 dB. Turbo codes can

DIFFERENCE-SET CYCLIC CODES

N	7	21	73	273	1057	4161
M	4	**10**	**28**	**82**	**244**	**730**
K	3	11	45	191	813	3431
d	4	**6**	**10**	**18**	34	66
k	3	**5**	**9**	17	33	65

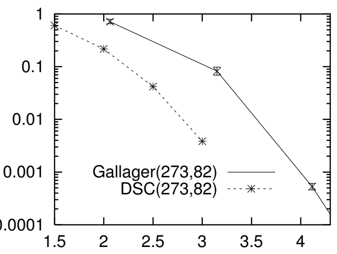
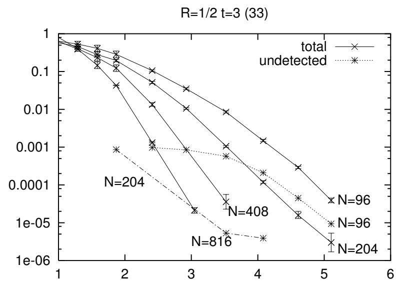


Figure 4. Difference-set cyclic codes – low–density parity–check codes satisfying many redundant constraints – outperform equivalent Gallager codes. The table shows the N, M, K, distance d, and row weight k of some difference-set cyclic codes, highlighting the codes that have large d/N, small k, and large N/M. All DSC codes satisfy N constraints of weight k. In the comparison the Gallager code had $(j,k) = (4,13)$, and rate identical to the DSC code. Vertical axis: block error probability; horizontal axis: E_b/N_0/dB. Data on DSC code performance provided by R. Lucas and M. Fossorier.

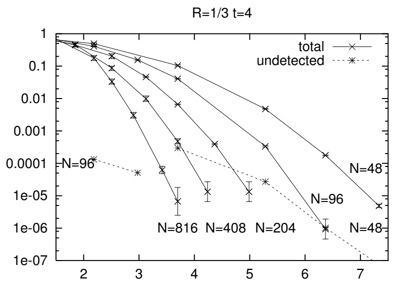
also be beaten by irregular Gallager codes defined over finite fields $GF(q)$ (Davey and MacKay 1998). However, these successes for Gallager codes have only been obtained by careful work, and it is notable how easily simple turbo codes and repeat-accumulate codes achieve almost as good results. The good performance of simple turbo codes and repeat-accumulate codes compared with simple Gallager codes can presumably be attributed to the use of state variables. It seems plausible therefore that the best codes will make use of state variables. Probably what is holding up the development of even better turbo codes is the need for a theory, comparable to the theory of irregular Gallager code design (Urbanke *et al.* 1999), for the construction of irregular graphs with state variables.

Codes with redundant constraints – 'the Tanner challenge'

The performance of Gallager codes can be enhanced by making a non-random code with **redundant sparse constraints** (Tanner 1981; Lucas *et al.* 1999). There is a difference-set cyclic code, for example, that has $N = 273$, and $K = 191$, but the code satisfies not $M = 82$ but N, *i.e.*, *273*, low-weight constraints (figure 4). It is impossible to make random Gallager codes that have anywhere near this much redundancy among their checks. The redundant checks allow the sum-product algorithm to work substantially better, as shown in figure 4, in which a DSC code outperforms a comparable regular binary Gallager code by about 0.7 dB. The (73,45) DSC code has been implemented on a chip by Karplus and Krit (1991) following a design of Tanner (1981). Product codes are another family of codes with redundant constraints.



(a)



(b)

Figure 5. Performance of Gallager codes with very small block lengths.
(a) Rate 1/2, column weight $j = 3$. Dotted lines show undetected errors occuring for blocklengths $N = 96$ and $N = 204$.
(b) Rate 1/3, column weight $j = 4$. Dotted lines show undetected errors occuring for blocklengths $N = 48$ and $N = 96$ only. Vertical axis: block error probability; horizontal axis: E_b/N_0/dB.

For example, the product with itself of a $(n, k) = (64, 57)$ Hamming code satisfying $m = 7$ constraints is a $(N, K) = (n^2, k^2) = (4096, 3249)$ code. The number of independent constraints is $M = 847$, but the sum-product decoder can make use of $2nm = 896$ equivalent constraints. Such codes have recently been named 'turbo product codes', but I think they should be called 'Tanner product codes', since they were first investigated by Tanner (1981). Product codes have the disadvantage, however, that their distance does not scale well with blocklength; the distance of a product code with blocklength n^2, built from two codes with distance d, is only d^2, so the ratio of distance to blocklength falls.

An open problem is to discover codes sharing the remarkable properties of the difference-set cyclic codes but with larger blocklengths and arbitrary rates. I call this task **the Tanner challenge**, in honour of Michael Tanner, who recognised the importance of such codes twenty years ago.

3 Do Gallager codes ever make undetected errors?

I use the term 'undetected error' to denote a decoding which returns a valid codeword not actually equal to the transmitted codeword. In our empirical experience with Gallager codes of many shapes and sizes, Gallager codes do not make undetected errors; their only failure mode is a 'detected error', that is, a decoding that fails to converge to a valid codeword, so that the recipient is aware that the block has been mis-received (MacKay and Neal 1996). Of course, we are decoding Gallager codes at noise levels well above half the minimum distance of the code, so, logically, undetected errors must have nonzero probability; nevertheless, they appear to be so rare as to be effectively nonexistent. Since this property is an important advantage of Gallager codes, we have explored empirically how far regular binary Gallager codes can be pushed before undetected errors show up. Figure 5 shows that undetected errors occur when we reduce the blocklength N to sufficiently small values – below 200 bits in the case of rate 1/3 codes with $j = 4$, and below 400 bits in the case of rate 1/2 codes with $j = 3$. The frequency of undetected errors appears to fall very rapidly with increasing blocklength and with increasing column weight j.

4 Stopping rules for the decoding of sparse graph codes

When we decode Gallager codes, we test the bit-by-bit best guess at each iteration to see if it is a valid codeword. If it is, then our decoder halts. Otherwise, the decoder continues, declaring a failure (a detected error) if some maximum number of iterations (*e.g.*, 200 or 1000) occurs without successful decoding.

In the turbo code and repeat-accumulate code community, a different decoding procedure is widely used. The decoding algorithm is run for a *fixed* number of iterations (irrespective of whether the decoder has actually settled in a consistent state at some earlier time), and performance curves are displayed as a function of the number of iterations. This practice is wasteful of computer time, and it blurs the distinction between undetected and detected errors. Undetected errors are of scientific interest because they reveal distance properties of a code. And in engineering practice, it would seem preferable for the detected errors to be labelled as erasures if practically possible – undetected errors are a great nuisance in many communication contexts.

I therefore demonstrate here that it is possible to detect convergence of the sum-product decoder of a repeat-accumulate code, just as in the case of turbo codes (Frey 1998). This assertion may be found confusing if the role of the decoder is viewed as 'finding the most probable state of the source bits'. Surely any hypothesis about the K source bits is a valid hypothesis, so how can we detect that the decoder has finished decoding? The answer is that decoding should be viewed as finding the most probable state of all the variables in the graph, not just the source bits. Early on in the decoding, there will be inconsistencies among the most probable states of internal variables in the graph. Only when a sensible decoding has been found will there be a state of harmony in the network. [Note that detecting convergence doesn't imply that we get rid of undetected errors; undetected errors will still occur if the decoder converges to an incorrect codeword.]

I used the following method to decide when to stop the decoding of a repeat-accumulate code.

THE STOP-WHEN-IT'S-DONE RULE FOR DECODING REPEAT-ACCUMULATE CODES.

1. While running the sum-product algorithm up and down the accumulator trellis, note the most probable state at each time. This state sequence – if you unaccumulate it – defines a sequence of guesses for the source bits, with each source bit being guessed q times, where q is the number of repetitions in the RA code.
2. When reading out the likelihoods from the trellis, and combining the q likelihoods for each of the source bits, compute the most probable state of each source bit. This is the state that maximizes the product of the likelihoods.
3. If *all* the guesses in (1) agree with the most probable state of the source bit found in (2) then the decoder has reached a valid state, so HALT. Otherwise continue.

The cost of these extra operations is small compared to the cost of decoding. Using this procedure, we can distinguish between undetected errors and detected errors in a repeat-accumulate code, as shown in the results of figure 2(b).

We can also evaluate how many iterations it takes to decode these codes, and quantify the potential savings from applying the above stopping rule. If, for example, the old-fashioned method runs all decodings for 20 iterations, the histogram in figure 6(b) shows that there would, for a particular code at 0.75 dB, be a block error probability of about 0.8 – roughly 80% of the decodings took more than 20 iterations. Since a small but substantial number took more than 40 iterations, you could run the old-fashioned decoder for twice as long, and the block error probability would fall by a factor of ten or so. However, using the stop-when-it's-done decoder, you can use roughly the same amount of computer time and get the error probability down by a factor of about one hundred.

Nothing is lost, because (if you log the stopping time in a file) you can always recover the old-fashioned graphs if anyone still wants them.

5 Empirical distribution of decoding times

We have investigated the number of iterations τ of the sum-product algorithm required to decode Gallager codes and repeat-accumulate codes. Given one code and a set of channel conditions the decoding time varies randomly from trial to trial. We find that the histogram of decoding times follows a power law, $P(\tau) \propto \tau^{-p}$, for large τ. The power p depends on the signal to noise ratio and becomes smaller (so that the distribution is more heavy-tailed) as the signal to noise ratio decreases.

(a)

(ii.b) (ii.c)

(iii.b) (iii.c)

Figure 6. Histograms of number of iterations to find a valid decoding for a repeat-accumulate code with source block length $K = 10000$ and transmitted block length $N = 30000$. (a) Block error probability versus signal to noise ratio for the RA code. (ii.b) $x/\sigma = 0.89$, $E_b/N_0 = 0.749$ dB. (ii.c) $x/\sigma = 0.90$, $E_b/N_0 = 0.846$ dB. (iii.b, iii.c): Fits of power laws to (ii.b) $(1/\tau^6)$ and (ii.c) $(1/\tau^9)$.

(a) (b)

Figure 7. (a) Histogram of number of iterations for an irregular binary Gallager code, rate 1/2, blocklength $N = 9972$, at $E_b/N_0 = 1.4\,\text{dB}$. (b) Log/log plot of iterations histogram showing that the tail of the distribution is well approximated by a power law. The straight line has slope -8.5. From MacKay *et al.* (1999).

We have observed power laws in repeat-accumulate codes and in irregular and regular Gallager codes. Figures 6(ii) and (iii) show the distribution of decoding times of a repeat-accumulate code at two different signal-to-noise ratios. The power laws extend over several orders of magnitude. Figure 7 shows the distribution of decoding times for an irregular Gallager code. The tail is well approximated by the power law $P(\tau) \sim \tau^{-8.5}$.

It would be interesting to understand the reason for these power laws.

References

Berrou, C., and Glavieux, A. (1996) Near optimum error correcting coding and decoding: Turbo-codes. *IEEE Transactions on Communications* **44**: 1261–1271.

Davey, M. C., and MacKay, D. J. C. (1998) Low density parity check codes over GF(q). In *Proceedings of the 1998 IEEE Information Theory Workshop*. IEEE.

Divsalar, D., Jin, H., and McEliece, R. J. (1998) Coding theorems for 'turbo–like' codes. In *Proceedings of the 36th Allerton Conference on Communication, Control, and Computing, Sept. 1998*, pp. 201–210, Monticello, Illinois. Allerton House.

Frey, B. J. (1998) *Graphical Models for Machine Learning and Digital Communication*. Cambridge MA.: MIT Press.

Gallager, R. G. (1962) Low density parity check codes. *IRE Trans. Info. Theory* **IT-8**: 21–28.

Gallager, R. G. (1963) *Low Density Parity Check Codes*. Number 21 in Research monograph series. Cambridge, Mass.: MIT Press.

Karplus, K., and Krit, H. (1991) A semi–systolic decoder for the PDSC–73 error–correcting code. *Discrete Applied Mathematics* **33**: 109–128.

Luby, M. G., Mitzenmacher, M., Shokrollahi, M. A., and Spielman, D. A. (1998) Improved low–density parity–check codes using irregular graphs and belief propagation. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, p. 117.

Lucas, R., Fossorier, M., Kou, Y., and Lin, S., (1999) Iterative decoding of one-step majority logic decodable codes based on belief propagation. Submitted.

MacKay, D. J. C. (1999) Good error correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory* **45** (2): 399–431.

MacKay, D. J. C., and Davey, M. C., (1998) Evaluation of Gallager codes for short block length and high rate applications. Available from `wol.ra.phy.cam.ac.uk/mackay/`.

MacKay, D. J. C., and Neal, R. M. (1996) Near Shannon limit performance of low density parity check codes. *Electronics Letters* **32** (18): 1645–1646. Reprinted *Electronics Letters*, **33**(6):457–458, March 1997.

MacKay, D. J. C., Wilson, S. T., and Davey, M. C. (1999) Comparison of constructions of irregular Gallager codes. *IEEE Transactions on Communications*. In press.

McEliece, R. J., MacKay, D. J. C., and Cheng, J.-F. (1998) Turbo decoding as an instance of Pearl's 'belief propagation' algorithm. *IEEE Journal on Selected Areas in Communications* **16** (2): 140–152.

Tanner, R. M. (1981) A recursive approach to low complexity codes. *IEEE Transactions on Information Theory* **27** (5): 533–547.

Urbanke, R., Richardson, T., and Shokrollahi, A., (1999) Design of provably good low density parity check codes. Submitted.